Hunting Down Malicious Code

The Reply Code Masters Team

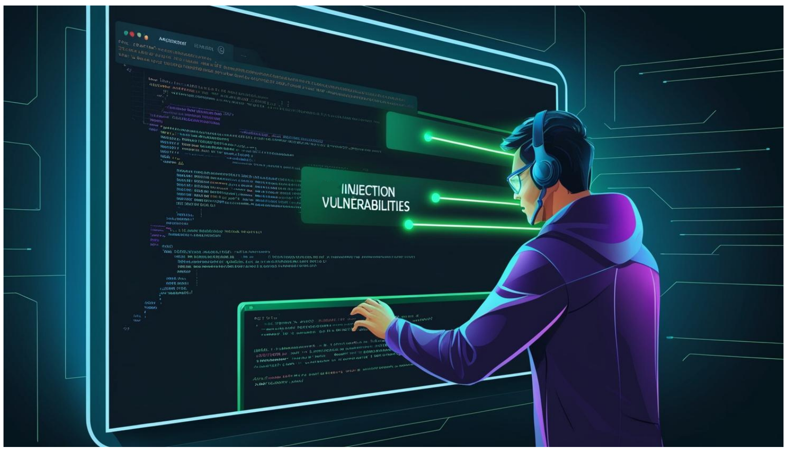
12 March 2025

1. Problem statement

Injections are one of the most insidious threats in the world of **cybersecurity**. They involve a class of exploits where an attacker inserts malicious code or commands into a system's input fields, aiming to compromise data integrity, gain unauthorized access or even take control of the entire system.

As the chief of security, it is your crucial duty to monitor system logs and detect any **intrusion attempts**. Every suspicious request could be hiding malicious code designed to exploit vulnerabilities in the software. Indicators of a potential attack include unusual special characters in input parameters, attempts to execute unexpected commands.

Analyzing system logs and leveraging threat detection tools is essential to prevent **irreversible damage**. Implementing input validation and sanitization mechanisms, monitoring suspicious activities and responding quickly to security alerts can mean the difference between a secure system and a catastrophic breach. Are you ready to identify and **neutralize threats** before it's too late?



You have a list of N lines of logs. Each line is an input provided by a user. You have to extract the basic commands from the logs. You can recognize a basic command because no other smaller log is included in the string.

2. Input format

The input file is a regular ASCII text file. Each line of the input file ends with a single $\backslash n$ character (UNIX-style). If a line contains multiple data, each value is separated by a single space character.

The first row of the input file will be composed of an integer number C, representing the number of test cases to be solved. The following rows represent for each test case C_i in order:

- The integer number N_i, indicating the number of provided logs
- N lines L_1, L_2, ..., L_N. Each line contains a string with different characters, indicating the log of a command compiled by users

3. Output format

The output file must be a regular ASCII text file. Each line of the output file must end with a single $\backslash n$ character (UNIX-style). The rows represent for each test case, in ascending order:

- a string indicating the test case number, in the format *Case #*C_i:
- an integer indicating the number of elegible logs
- a line with L_0, L_1, ..., L_{N-1} logs satisfying the condition

4. Constraints

For each case:

- At least one eligible log L_i satisfies the condition.
- Maximum length of each log: $1 \leq \text{length}(L_i) \leq 250$.
- Each log L_i appears once in the input file.

For every input:

- **Input 1** : $C = 1$, $N \leq 15$
- **Input 2** : $C = 5$, $N \leq 5000$
- **Input 3** : $C = 1$, $N \leq 40000$
- **Input 4** : $C = 1$, $N \leq 50000$

5. Example

5.1. Input file example

```
1
5
find_user
deletetable
find_user:alice;send_promo
username="admin';deletetable=users;--"
#print
```

The Case 1 to be solved contains 5 logs. Only 3 of these logs satisfy the condition where no other smaller log is included.

5.2. Output file example

```
Case #1: 3 find_user deletetable #print
```

For the case C_1, the solution is composed by 3 lines: log n.1, log n.2, log n.5

- *`find_user:alice;send_promo`* is not a basic command, since it contains the *find_user* log
- *`username="admin';deletetable=users;--"`* is not a basic command, since it contains the *deletetable* log

6. CTF recall

The zoo keeper's sloppy security reminds you of past injection exploits—hiding secrets in plain sight but failing to secure them properly. Just like those misconfigured inputs in the logs, his mistakes could be the key. Maybe revisiting that CTF challenge will reveal a similar vulnerability waiting to be exploited.